Multi-Processing CTH: Porting Legacy FORTRAN Code to MP Hardware (U)

R. L. Bell, M. G. Elrick, and E. S. Hertel, Jr.

Sandia National Laboratories

Abstract

CTH is a family of codes developed at Sandia National Laboratories for use in modeling complex multi-dimensional, multi-material problems that are characterized by large deformations and/or strong shocks. A two-step, second-order accurate Eulerian solution algorithm is used to solve the mass, momentum, and energy conservation equations. CTH has historically been run on systems where the data are directly accessible to the cpu, such as workstations and vector supercomputers. Multiple cpus can be used if all data are accessible to all cpus. This is accomplished by placing compiler directives or subroutine calls within the source code. The CTH team has implemented this scheme for Cray shared memory machines under the Unicos operating system. This technique is effective, but difficult to port to other (similar) shared memory architectures because each vendor has a different format of directives or subroutine calls. (U)

A different model of high performance computing is one where many (>1000) cpus work on a portion of the entire problem and communicate by passing messages that contain boundary data. Most, if not all, codes that run effectively on parallel hardware were written with a parallel computing paradigm in mind. Modifying an existing code written for serial nodes poses a significantly different set of challenges that will be discussed. CTH, a legacy FORTRAN code, has been modified to allow for solutions on distributed memory parallel computers such as the IBM SP2, the Intel Paragon, Cray T3D, or a network of workstations. (U)

The message passing version of CTH will be discussed and example calculations will be presented along with performance data. Current timing studies indicate that CTH is 2-3 times faster than equivalent C++ code written specifically for parallel hardware. CTH on the Intel Paragon exhibits linear speed up with problems that are scaled (constant problem size per node) for the number of parallel nodes. (U)

Introduction

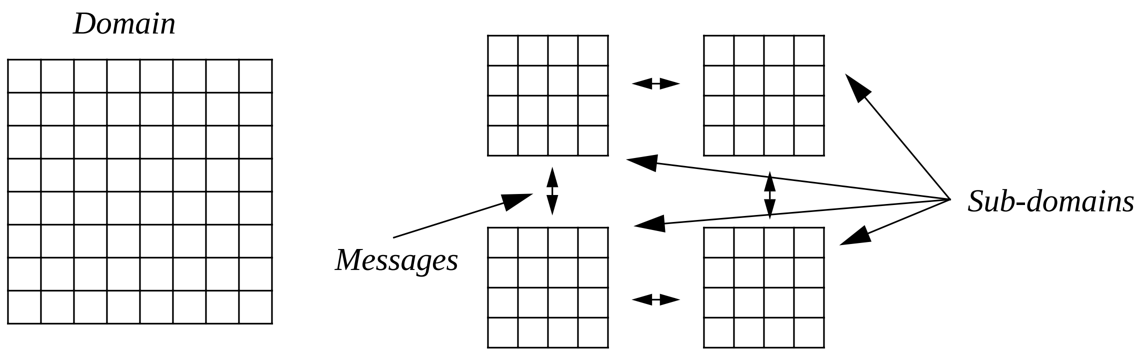
The future of high performance computing is directed at scalable parallel computers where problems are solved by breaking a large domain into many small sub-domains. Sandia has been working in the development of tools (Robinson, 1992 and McGlaun, 1995) for massively parallel (MP) computers for several years. Up to this time the emphasis has been on the development of these tools from scratch, not in the porting of existing codes to MP machines. PCTH (Robinson, 1992) was developed to solve the conservation equations of mass, momentum, and energy for shock physics. In this case, the algorithms used in CTH (McGlaun, 1990) were reprogrammed using C++ and a message passing concept for the parallelization. This effort proved to be very successful in that we demonstrated effective use of MP computers. For a variety of reasons, we decided to take the knowledge gained in the development of PCTH and port CTH to MP computers. This paper is a description of the techniques used to port CTH to MP computers. (U)

Assumptions and Background

We assumed that CTH would continue to be used on a variety of computing platforms. We currently support CTH on all major Unix workstations and Cray vector computers. This requirement meant that the core functionality of CTH could not be compromised and the changes necessary to port CTH to MP computers must co-exist with the serial code. Furthermore, we required that the MP code give identical results regardless of number of nodes used to solve the problem. The emphasis for MP was (and is) three-dimensional simulations, but we would port all geometry options to MP computers. We would implement machine specific changes in a way that supported portability in message passing interfaces. We would port all possible numerical features and physical models to MP computers that were cast as “local” to each computational cell. Local in this sense means that the solution can be advanced one time step based on information in the (local) computational cell or in its neighboring cells. This locality restriction affects only one model in the current version of CTH. Given these restrictions and requirements, we initiated a program to port CTH to MP computers. (U)

Distributed data MP computers are characterized by a number (generally large, Sandia’s Intel Paragon has ~1800 nodes and DOE’s ASCI Red Machine has ~9000 nodes) of discrete computational nodes consisting of memory, a commodity cpu chip, and access to an internal communications network. One computing technique that can be employed on this machine is referred to as single program multiple data (SPMD). It is referred to SPMD because the same executable is running on each computational node, but each executable is working with a different data set. Algorithms that depend on a fixed logically connected mesh are relatively simple to map onto a SPMD machine. The technique used for CTH is similar to that used for PCTH in that the entire problem domain is broken up into sub-domains that reside on individual computational nodes. Communication between nodes (each containing separate regions of global mesh) are handled by the use of “ghost cells” and explicit messages that CTH passes between nodes. The use of “ghost cells” is a typical technique for applying boundary conditions in Eulerian codes. The “ghost cells” allow for the finite-difference equations to be independent of edges and corners. For an external boundary, the ghost cell data are based on the selected boundary condition approximation. For an internal boundary, the ghost cells contain real data that was acquired in a message passed from a neighboring node. A simple example of mesh decomposition is displayed in Figure 1. (U)

Figure 1: CTH Mesh Decomposition Scheme



CTH Solution Sequence and Message Passing Process

CTH is a family of several codes that work together to solve shock physics problems. For the purposes of this paper only two of the codes need to be considered, CTHGEN and CTH. CTHGEN reads the user input and builds a time-zero representation of the problem specifications. CTH reads additional user input and the time-zero data from CTHGEN and initiates the time integration process. Very little of this process changes for MP computers. The copy of CTHGEN that runs on node 0 reads the user input and broadcasts it to all other nodes. Given the total number of nodes requested by the user and the total problem size, CTHGEN decides how to map the problem space onto the nodes. The algorithm it uses to build the sub-domains is to equalize the amount of work on each node (equal number of cells per node) and to keep each sub-domain as close to cubic as possible. The requirement for cubic sub-domains is for two reasons, minimize the surface to volume ratio and to make the message data volume as equal as possible. Once the problem has been mapped onto the available nodes, the individual copies of CTHGEN insert material in the respective sub-domains, define material properties, and complete the time-zero representation of the problem specifications (“building the database” is the typical nomenclature). CTH starts in much the same way, the copy of CTH that runs on node 0 reads the user input and broadcasts it to all other nodes. Once the broadcast is complete, each node reads its individual database file. At this point, the time integration starts. (U)

The solution sequence for CTH has not been changed for MP computing. Essentially every time the ghost cell values have been changed, CTH must exchange these new values with neighboring nodes before the updated values are used in the solution sequence. The overall solution sequence of CTH is a Lagrangian step followed by a remap and then a database modification step where materials may be discarded or velocity transformations applied. The Lagrangian step consists of several tasks. The first is where the artificial viscosity is defined, during this task messages are exchanged with neighbors to calculate the correct boundary values. Once the artificial viscosity and pressure are defined, CTH calculates the new cell velocities. The new cell velocities are exchanged with neighboring nodes. The new cell velocities are used to update the stress deviators at this time and then the new stress deviators are exchanged with neighboring nodes. The new stress deviator information leads to new energy terms from the PdV work. After the energy is updated, the Lagrangian step is effectively complete. Special models like the multi-phase reactive flow package perform tasks to prepare for the remap step. At the end of the Lagrangian step, all ghost cell values are exchanged for the last (fourth) time. (U)

CTH uses a second order accurate advection scheme. This scheme, based on work by van Leer (1977), determines a linear slope across each “donor” cell. To calculate this slope, data from three cells are required, the donor cell, the cell upstream, and the cell downstream. A given node “knows” the values in the ghost cell, the first real cell, and the next cell near a nodal boundary. If the flow is “outward” (from the last real cell into the ghost cell) the code has enough information to calculate the slope across the donor cell (the last real cell). However, if flow is from the ghost cell into the first real cell, the code does not have information about the “upstream” cell (the cell beyond the ghost cell). But, if a node shares this boundary, the inflow for this node is exactly the same as the “outflow” for the adjacent node. So we have developed a new set of subroutines which calculate “outflow” values for each node boundary. These values are collected and passed to the adjacent nodes. If a node calculates an “inflow” value and finds that another node shares that boundary, the value from the message is used rather than the incorrectly calculated value. By using the second order accurate “outflow” value from the adjacent node we are able to duplicate results from single-node simulations with MP simulations. (U)

Several exchanges must be completed during the Eulerian (or remap) step. Each time ghost cell values are modified, data must be exchanged with neighboring nodes. The first task in the Eulerian driver converts volume fraction values to volumes, new ghost cell data must be exchanged with neighbor nodes. CTH uses an operator split scheme for the remap step. Each time the remap is completed in a particular direction, the ghost cell values must be exchanged with neighboring nodes. During this step momenta and updated mass values must also be exchanged because CTH uses the half index shifted momentum advection scheme of Benson (1991). This method requires correct velocities at the node mesh boundaries. Since velocities on the edges of isolated material cells are also modified during the remap step, these corrected velocities must also be exchanged. Finally, after all remap steps have been completed, the volumes are converted back to volume fractions and the Eulerian energy balance is accomplished. This step calls the equation of state for each material yielding new cell pressures, temperatures, and sound speeds. One of the last steps in the remap is to calculate the minimum time step, this is first done for each node and then a global minimum is done to determine the time step for the next computational cycle. This is the last time that significantly sized messages are exchanged. (U)

A significant additional feature that needs to be addressed is the use of Lagrangian or tracer data points. The code records data at tracer locations as the simulation progresses. The points can either move with the bulk flow field or be fixed in space. Each tracer point is initially placed in the mesh based on the user supplied coordinates. If the tracer coordinates are in the interior of the volume of space “owned” by a particular node, these coordinates are recorded by that node and a flag is set in the tracer data storage. This tracer’s coordinates in all other nodes will be recorded as (1.0e20, 1.0e20, 1.0e20), the upper right hand coordinate of the universe. Further, the location flag will be set to indicate non-ownership by that node. This allows a quick check on whether or not a particular tracer is active in a given node. At the end of each cycle, the tracer positions (coordinates) are updated by the nodes “owning” the tracers. Messages are then exchanged between nearest neighbor nodes. After all six messages have been exchanged, all 27 nodes surrounding the actual position of the tracer know the true coordinates. These coordinates are then compared with the limiting coordinates for each node. If the tracer has migrated to a new node, the new coordinates are set in the tracer array and the tracer flag is reset to indicate ownership. The tracer coordinates for all other nodes will be set to (1.0e20, 1.0e20, 1.0e20). There is no need to propagate the coordinates to nodes beyond nearest neighbors since the time step controls prevent any tracer moving more than one cell width in any given time step. (U)

For a three-dimensional calculation, 24 of what we characterize as large messages are passed during the Lagrangian step, 48 large messages are also passed during the Eulerian step. A large message contains all cell variables on the face adjoining two nodes. For Sandia’s Intel Paragon, available node memory limits each node to sub-domains of ~24^3. Typical problems consist of 40-80 variables per cell, therefore large messages are on the order of 200-400 kbytes. Several small messages are also passed during the solution sequence. These messages are typically passed during the calculation of global sum and minimization processes. (U)

CTH MP Scaling Results

The serial version of CTH has been modified to allow for single program multiple data computing on parallel computers. To date, CTH has been tested on networks of workstations using the PVM library, an IBM SP2 using the MPI Library, the Intel Paragon running both SUNMOS (a Sandia/UNM developed Operating System) and OSF using the native message passing library (known as NX), PVM, and MPI, and a small scale (60 nodes) prototype of the ACSI Red machine (also known as the Intel TeraFlop) using NX. The most extensive testing has been done on the Paragon and the results displayed for this paper are from the Paragon but should be typical for all of the machines described above. (U)

There are two ways of measuring performance on parallel computers, the first is to take a fixed problem size and monitor the run time as a function of the number of compute nodes, the second is to keep the problem size per node fixed and monitor the run time as a function of compute nodes. The first method has a natural limit in that as the work per node decreases, an asymptotic limit will be reached when the minimum number of cells per node (for CTH it is 27 for three-dimensional geometries) occurs. It is a useful measure in that it gives information about start-to-completion time speed up for a fixed simulation. Table 1 displays data taken from the Paragon for a two material single point initiation problem run in two-dimensional cylindrical geometry. MP computers suffer from a start-up penalty as the input is broadcast and initialization files are read. The data in Table 1 has this start-up time removed for the calculation of the grind time. Hydrocodes have used a performance metric known as grind time for a number of years, where grind time is defined as the amount of cpu time necessary to complete all calculations on a single cell for a single time step. For three-dimensional simulations, high-end serial workstations and single-cpu vector supercomputers have grind times on the order of 100 μs/zone-cycle. From the data in

Table 1: CTH Fixed Problem Size Performance on MP Computers

Nodes	CPU Time (s)	Grind Time (μs/zone-cycle)	Cell per Node
4	1690	92.1	45,000
8	877	47.2	22,500
16	482	26.2	11,250
32	263	14.4	5,625
64	150	8.4	2,812
128	86	4.6	1,406
256	52	2.6	703
512	32	1.7	351

Table 1 it can be clearly seen that an asymptote is being reached as the work per node decreases and the run times are dominated by the message passing. The second method for measuring performance is critical to proving a scalable decrease in grind time for large numbers of nodes. For this test, the total problem size increases at the same rate as the increase in the number of nodes. Table 2 displays data taken from the Paragon for a three material explosively formed projectile problem run in three-dimensional rectangular geometry. Several points need to be noted from an

Table 2: CTH Scaled Problem Size Performance on MP Computers

Nodes	CPU Time (s)	Grind Time (μs/zone-cycle)	Cell per Node
2	4127	319	27648
4	4639	179	27648
8	5287	102	27648
16	5193	50.2	27648
32	5152	24.9	27648
64	5119	12.4	27648
128	5222	6.3	27648
256	5503	3.3	27648
512	5307	1.3	27648
1024	-	0.57	13824

examination of Table 2. After the initial start-up time is overcome, the cpu time to complete the simulation is relatively constant. Furthermore, the start-up time is dominated by IO from parallel disks and across the inter-node network which depends heavily on the instantaneous machine loading. The internal grind time calculations do not include the start-up times and are a more accurate indication of performance. All simulations were run as interactive jobs and some variation is to be expected due to varying system loads. From 8 thru 256 nodes, the grind time decreases by a factor of 31.2 as the number of nodes increases by a factor 32 indicating a linear scaling of performance. Due to constraints in node memory on Sandia's Paragon, the number of cells per node was decreased to half of the previous simulations for the 1024 node problem. Other tests on the entire machine (~1800 nodes) typically give grind times of 300 to 600 nanoseconds. Comparative tests with PCTH and CTH on identical problems show that CTH is 3 to 5 times faster than PCTH. We attributed the bulk of this speed up to be due to the superior performance (that is, maturity) of Fortran versus C++ compilers. (U)

Conclusions

We have successfully integrated message passing functions into the serial version of CTH. We drew upon the knowledge gained through the development of PCTH to implement these changes in CTH and believe that it would have been extremely difficult to modify CTH for message passing without the trail-blazing work of Robinson (1992). We have shown scaled speed-up on the Intel Paragon through 1800 nodes and also demonstrated the ability to maintain serial and parallel code constructs in the same source code. If we extrapolate the Paragon performance to the ASCI Red machine, we expect grind times to be below 10 nanoseconds per zone-cycle. (U)

References

Benson DJ, (1991) *Momentum Advection on a Staggered Mesh*. J. Comp. Phys. 100:1

McGlaun JM, Thompson SL, Kmetyk LN, Elrick MG, (1990) *CTH: A Three-Dimensional Shock Wave Physics Code.* Int. J. Impact Engng. 10:351

McGlaun JM, Robinson AC, and Peery JS, (1995) *The Development and Application of Massively Parallel Solid Mechanics Codes,* 1995 International Conference on Computational Engineering Science, Mauna Lani, HI

Robinson AC, Ames AL, Fang HE, Pavlakos C, Vaughan CT, Campbell P, (1992), *Massively Parallel Computing, C++ and Hydrocode Algorithms.* Proceeding of the 8th Conference in Computing in Civil Engineering, Dallas, TX.

van Leer B, (1977) *Towards the Ultimate Conservative Difference Scheme IV. A New Approach to Numerical Convection.* J. Comp. Phys. 23:276

Acknowledgment

This work performed at Sandia National Laboratories supported by the U. S. Department of Energy under contract #DE-AC04-76DP000789.

The authors would also like to thank Elliot Fang and Marlin Kipp for their help in running simulations that are discussed in this paper.